VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Aplikace pro záznam a vizualizaci dat senzorů na platformě Android

Application for Sensor Data Recording and Visualisation on Android Platform

2018 Marek Ulip

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Marek Ulip**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Aplikace pro záznam a vizualizaci dat senzorů na platformě Android
Application for Sensor Data Recording and Visualisation on Android Platform

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit aplikaci pro periodický záznam dat senzorů, poskytovaných platformou Android, a dat z externího (externích) Bluetooth Smart zařízení. Platforma bude umožňovat vizualizaci uložených dat ve 2D a 3D, a jejich export včetně polohy do vhodného výměnného formátu (např. GPX), doplněného o dodatečné informace.

1. Prostudujte existující aplikace podobného zaměření a zjistěte, jaké možnosti poskytují a jak data zaznamenávají.
2. Nastudujte možnosti vizualizace měřených dat a možnosti jejich záznamu.
3. Vyberte vhodný externí Bluetooth Smart senzor, zvolte implementační prostředí, popište knihovny, použité pro implementaci aplikace, a jejich výběr zdůvodněte.
4. Analyzujte a navrhněte aplikaci.
5. Implementujte řešení pro záznam dat ze senzorů a jejich vizualizaci pomocí OpenGL ES.
6. Aplikaci otestujte alespoň na dvou zařízeních, a pokud to bude možné, vhodným způsobem publikujte. Výsledky testů vyhodnoťte.

Seznam doporučené odborné literatury:

[1] Steele, J., To, N.: The Android Developer's Cookbook: Building Applications with the Android SDK, Addison-Wesley Professional, 2010, ISBN-13: 978-0321741233
[2] Meier, R.: Professional Android 4 Application Development, Wrox, 2012, ISBN-13: 978-1118102275[1] Steele, J., To, N.: The Android Developer's Cookbook: Building Applications with the Android SDK, Addison-Wesley Professional, 2010, ISBN-13: 978-0321741233
[3] Gabriel Svennerberg, Cameron Turner: Beginning Google Maps API 3. Apress, 2010, 328 stran, ISBN: 978-1-4302-2802-8.
[4] SimpleLink™ SensorTag Software [online] [cit. 2016-02-15] Dostupné z <http://www.ti.com/tool/sensortag-sw?keyMatch=SensorTag&tisearch=Search-EN-TechDocs>

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Moravec, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 24. dubna 2018 .....

Rád bych na tomto místě poděkoval Ing. Pavlu Moravcovi, PhD. za odbornou pomoc a konzultace při vytváření této bakalářské práce.

Abstrakt

Cílem této bakalářské práce bylo vytvoření aplikace pro periodický záznam dat ze senzorů pro platformu Android, a dat z externího Bluetooth Smart zařízení. Mobilní zařízení, které je poháněno platformou Android a disponuje touto aplikací, je schopno snímat data ze senzorů spolu s geolokací a následně je exportovat ve formátu GPX. Práce pojednává také o základních informacích ohledně platformy Android, vlastnostech senzorů a o technologii Bluetooth.

Klíčová slova: Android, aplikace, senzory, záznam senzorových dat, Bluetooth Low Energy, Bluetooth Smart, GPX

Abstract

The aim of the bachelor thesis was to create an application for periodic recording of data from sensors for Android platform, and data from an external Bluetooth Smart device. An Android-powered mobile device with this app is able to capture sensor data along with geolocation and afterwards export it to GPX format. The thesis also discusses basic information about Android platform, sensor properties and, about Bluetooth technology.

Key Words: Android, application, sensors, recording of sensor data, Bluetooth Low Energy, Bluetooth Smart, GPX

Obsah

Seznam použitých zkratek a symbolů

BT	–	Bluetooth
BLE	–	Bluetooth Low Energy
GPS	–	Global Positioning System
GPX	–	GPS Exchange Format
MEMS	–	Microelectromechanical systems
CCD	–	Charge-coupled Device
LED	–	Light-Emitting Diode
GAP	–	Generic Access Profile
GATT	–	Generic Attribute Profile
UUID	–	Universally Unique Identifier
JSON	–	JavaScript Object Notation
XML	–	eXtensible Markup Language
URI	–	Unified Resource Identificator
CRUD	–	Create Read Update Delete
OpenGL	–	Open Graphics Library
API	–	Application Program Interface

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Obsahem této bakalářské práce je popis implementace aplikace jménem Droidsor, která slouží pro zobrazení, záznam a export dat ze senzorů, které mohou být součástí chytrého telefonu, nebo součástí externího zařízení SensorTag CC2650. V aplikaci by mělo být možné vybrat senzory, které budou zaznamenávány a nastavit, jak často budou údaje ze senzorů získávány. Aplikace by také měla umožňovat získávat během záznamu pozici GPS, taktéž s nastavitelnou frekvencí. Exportovaná data by měla být ve formátu GPX.

Motivací pro vytvoření aplikace Droidsor byla skutečnost, že aplikací, které zaznamenávají data ze senzorů, ať už interních, nebo externích, není mnoho. A z těchto aplikací jenom málokterá umožňuje export zaznamenaných dat v jakékoliv podobě a při vývoji Droidsoru nebyla nalezena jediná, která by spolu s daty zaznamenávala i GPS pozici a umožňovala exportovat data ve formátu, který GPS pozici podporuje.

Ve 2. kapitole je probrán princip, na kterém fungují jednotlivé senzory, dále pak komunikace s externími senzory, kde lze nalézt stručný popis technologie Bluetooth a Bluetooth Low Energy a nakonec je zmíněn senzor SensorTag CC2650, který je využit při této práci. 3. kapitola se zabývá stručným popisem platformy Android, jeho funkcí a frameworků, použitých při tvorbě této práce. 4. kapitola uvádí možnosti, kterými je možné provést vizualizaci a záznam dat a v poslední 5. kapitole teoretické části jsou uvedeny aplikace s podobným zaměřením a srovnání těchto aplikací.

V praktické části je nejprve v kapitole 6 provedena analýza, která pojednává o výběru verzí Android, na které implementovaná aplikace cílí, popisuje návrh a význam služby použité v Droidsoru, návrh databáze a nakonec uvádí hlavní prvky uživatelského rozhraní. Kapitola 7 popisuje implementaci klíčových tříd a funkcí se zaměřením na zajímavé nebo komplikované části. Poslední 8. kapitola se zabývá testováním se zaměřením na bezchybný běh aplikace, rychlost náročných operací, spotřebu baterie a objemu uložených dat.

2 Senzory

V této kapitole je popsán princip senzorů, které mohou být součástí mobilních zařízení nebo externího zařízení SensorTag CC2650. Dále kapitola probírá samotný SensorTag CC2650 a technologie použité pro komunikaci s tímto senzorem.

2.1 Charakteristika senzorů

Senzory v mobilních zařízeních někdy využívají jiný způsob měření pro získání dat ze svého okolí, než externí senzory. Jelikož by byl popis všech principů rozsáhlý, zaměřuje se tato kapitola pouze na principy nejčastěji využívané anebo ty, kterých je využito u zařízení SensorTag CC2650. Ostatní jsou pouze zmíněny.

2.1.1 Pohybový senzor

Součástí pohybového čidla může být více senzorů. Pohybové čidlo zařízení SensorTag CC2650 spojuje dohromady akcelerometr, gyroskop a magnetometr.

Akcelerometr – Akcelerometr [1] slouží pro měření zrychlení. Z mnoha druhů akcelerometrů je u mobilních zařízení nejčastěji využíván MEMS, který je navržen tak, aby zaznamenal změnu z konstantní nebo nulové rychlosti. Při změně dochází k vibracím, spojených tímto pohybem. Akcelerometr využívá mikroskopické krystaly, na kterých se při působení vibrací tvoří napětí, odpovídající danému zrychlení. Tomu se říká piezoelektrický jev, díky kterému je možné určit směr gravitace a tím i natočení přístroje.

Další druhy akcelerometru jsou mechanické, které využívají pružinku, kde zrychlení je měřeno délkou natažení pružinky při pohybu, a kapacitní, které využívají kondenzátoru, kde změna vzdálenosti mezi dvěma ocelovými pláty způsobí změnu kapacity a tu lze použít pro změření síly, která působila.

Gyroskop – K akcelerometru je často tento senzor připojen. Slouží podobně jako akcelerometr s tím rozdílem, že gyroskop měří úhlovou rychlost, díky čemuž se dá určit naklonění a natočení zařízení. V kombinaci s akcelerometrem lze přesněji určit skutečný pohyb zařízení v prostoru.

Magnetometr – Tento senzor je využíván jako kompas. Také je použit i GPS navigací, díky čemuž dochází k rychlejšímu určení polohy. Magnetometr bývá tvořen zmagnetizovaným kouskem plechu, u kterého dochází ke změně odporu vlivem magnetického pole. Změna odporu se vyhodnocuje měřením v uzavřeném elektrickém obvodu.

2.1.2 Senzor vlhkosti [2]

Z několika možných způsobů měření využívá SensorTag CC2650 kapacitní senzor vlhkosti. Tento senzor měří relativní vlhkost umístěním tenkého proužku oxidu kovu mezi dvě elektrody. Elektrická kapacita oxidu kovu se mění spolu s relativní vlhkostí atmosféry.

Další druhy jsou rezistentní, které využívají ionty v soli k měření elektrické impedance atomů a termální, kdy dva termální senzory vytvářejí elektřinu, založenou na vlhkosti okolního vzduchu.

2.1.3 Tlakový senzor [3]

Tlakové senzory v mobilních zařízeních měří tlak pomocí piezoelektrického článku. Obsahuje kolem 40 krystalických materiálů, které při napnutí generují elektrický náboj. Pnutí způsobené namáháním membrány je přeměněno na elektrický náboj, který je úměrný tlaku.

Další druhy barometru měří tlak z rozdílu odporu (využití rezistoru), rozdílu kapacity (využití kondenzátoru), rozdílu indukce (využití magnetického obvodu), oscilace (využití vibrujícího prvku), rozdíl z intenzity světla (využití LED a iontového rozdílu).

Barometr v zařízení SensorTag CC2650 měří absolutní tlak, což je tlak ve vztahu k perfektnímu vakuu. Perfektní vakuum je stav, kdy v atmosféře není přítomna žádná hmota, a tudíž je v prostředí nulový tlak.

2.1.4 Senzor okolního světla [4]

Existuje více druhů tohoto senzoru a každý z nich funguje lehce odlišným způsobem. Například fotobuňka nebo fotorezistor mění svůj odpor, když na ně svítí světlo, čímž se určuje jeho intenzita. CCD snímač přenáší nabité signály. Fotonásobiče detekují světlo a násobí ho.

2.1.5 Tepelný senzor [5]

Tepelný senzor zařízení SensorTag CC2650 využívá termočlánku. Tento senzor je vyroben spojením dvou odlišných kovů. Jeden konec se anglicky označuje jako “Hot junction”. Druhý konec těchto odlišných kovů se označuje jako “Cold end” nebo “Cold junction”. Cold junction je vytvořen na nejvzdálenějším bodu termočlánkového materiálu. Pokud dojde k rozdílu v teplotách mezi hot junction a cold junction, vytvoří se malé napětí. Toto napětí může být změřené a následně použito k určení teploty.

Kromě termočlánku existují ještě odporové snímače teploty (anglicky resistance temperature device), které měří teplotu na základě změny odporu a termistory, které využívají stejný princip jako odporové snímače teploty, ale používají polovodičové součástky a díky tomu jsou přesnější.

2.2 Komunikace s externím senzorem

Pro komunikaci s externím senzorem se využívá Bluetooth. Od verze 4.0 byla tato technologie rozdělena na tři části, a to Bluetooth High Speed, Bluetooth Classic a Bluetooth Low Energy.

Bluetooth High Speed kombinuje Bluetooth a WiFi, aby dosáhl vysokých přenosových rychlostí, ale vyžaduje čip, kombinující Bluetooth a WiFi, který je poměrně drahý.

Aktuální verze Bluetooth 5.0, která byla představena 16. června 2016, se zaměřuje hlavně na internet věcí (IoT) a většina změn se týká BLE, z nichž hlavní jsou zvýšení rychlosti přenosu dat a dosahu.

2.2.1 Bluetooth Classic

Bluetooth Classic je navržený pro nepřetržitý obousměrný přenos dat s vysokou propustností. Je vysoce efektivní, ale pouze na krátké vzdálenosti, a proto je to ideální řešení pro vysílání audia a videa, pro připojení k myši nebo jiným zařízením, které potřebují soustavný širokopásmý spoj.

2.2.2 Bluetooth Low Energy

Bluetooth Low Energy poskytuje aplikační propustnost pouze 0.3 Mbps. Data jsou posílána v malých 20 bytových balíčcích, ale dosah může být více jak 100 metrů a minimální zpoždění mezi nepřipojeným stavem a přenosem dat může být v řádu milisekund, zatímco Bluetooth Classic má zpoždění okolo 100 milisekund.

Hlavní výhodou je nízká spotřeba energie. Ačkoliv maximální proud BLE (až 15 mA) je polovina proudu BT Classic (až 30 mA), může být spotřeba BLE zařízení až 100krát menší.

2.2.3 Porovnání Bluetooth Classic a BLE

Spotřeba energie – Nízká spotřeba energie technologie BLE umožňuje malým zařízením zůstat v provozu po dobu 5 – 10 let. Zařízení BLE je po většinu času v režimu spánku a zapíná se pouze, když je zahájena komunikace. Pro srovnání – čip bezdrátové technologie BLE vysílající rychlostí 1 Mbps na vzdálenost pár metrů spotřebuje pouze desetinu energie v porovnáním s čipem Bluetooth Classic přenášejícího stejná data na stejnou vzdálenost.

Propustnost – Skutečná přenosová rychlost u BLE je kolem 100 – 250 Kbps v porovnání se zhruba 2 Mbps u Bluetooth Classic.

Cena – BLE zařízení jsou levnější ve srovnání s Bluetooth Classic. To je způsobeno tím, že Bluetooth Classic bylo navrženo pro použití v široké škále aplikací, což se bez kompromisů neobešlo.

Počet připojených zařízení typu slave – BLE zařízení může obsloužit větší množství zařízení. Počet zařízení je ovlivněn implementací a dostupnou pamětí.

Rychlost připojení – Navázat spojení se zařízením BLE je mnohem rychlejší v porovnání s Bluetooth Classic.

Z porovnání lze vyčíst, že technologie BLE je vhodná pro zařízení, která posílají malé množství dat, požadují dlouhou výdrž a jsou levná. Tyto požadavky splňuje i zařízení SensorTag CC2650.

2.2.4 Základní pojmy Bluetooth Low Energy

Pro lepší pochopení popisu implementace Droidsoru je zde třeba uvést několik pojmů, se kterými BLE technologie pracuje.

O navázaní spojení se stará GAP [6]. GAP definuje různé role pro zařízení, ale základní koncepty jsou centrální zařízení a periferní zařízení.

Periferní zařízení jsou malá, nízko spotřební s omezenými zdroji, které se mohou připojit k mnohem výkonnějšímu centrálnímu zařízení.

Centrální zařízení jsou většinou mobilní telefony nebo tablety, ke kterým se periferní zařízení připojují.

Po navázání spojení pomocí GAP je možné přenášet data. Generic Attribute Profile (GATT) [7] využívá konceptů služeb (angl. Services) a charakteristik (angl. Charateristics) pro přesnost dat mezi zařízeními. Pro uchování služeb, charakteristik a souvisejících dat je využit obecný datový protokol Attribute Protocol (ATT). U GATT je periferní zařízení označeno jako GATT server, které drží ATT data a definice služeb a charakteristik. Jako GATT klient je pak označováno centrální zařízení, které posílá požadavky GATT serveru.

GATT transakce jsou založeny na vnořených objektech nazývaných Profily, služby a charakteristiky.

Profily na samotném zařízení nejsou přítomny, je to kolekce služeb, zkompilovaných buď Bluetooth SIG anebo výrobcem periferních zařízení.

Služby jsou použity k rozdělení dat na logické celky a obsahují konkrétní kousky dat, nazývané charakteristiky. Služba může mít jednu nebo více charakteristik a každá služba se od ostatních odlišuje pomocí unikátního ID, zvaného UUID, které může být buď 16 bitové (pro oficiální BLE služby) anebo 128 bitové (pro vlastní služby).

Charakteristiky zapouzdřují jediný datový bod, i když může obsahovat i pole relativních dat, jako například údaje z os akcelerometru X/Y/Z. Stejně jako služby, tak i charakteristiky se od sebe odlišují pomocí předdefinovaných 16 nebo 128 bitových UUID. Právě charakteristiky jsou použity pro interakci s periferním BLE zařízením, jelikož je možné data do charakteristiky zapisovat.

Deskriptor je součástí charakteristiky a je to definovaný atribut, který popisuje určitou hodnotu z charakteristiky. Například u zařízení SensorTag CC2650 je využit k aktivaci periodického zasílání dat ze senzorů.

2.3 SensorTag CC2650

Sada vyhodnocovací soupravy CC2650 SensorTag funguje jako periferní BLE zařízení, založené na multifunkční CC2650 (bezdrátové MCU platformě), zahrnující pět periferních senzorů s kompletním softwarovým řešením pro ovladače senzorů, propojených se serverem GATT, který běží

Obrázek 1: SenzorTag CC2650 [9]

na TI BLE-Stack v2. GATT server obsahuje primární služby pro nastavení a datové kolekce každého senzoru. [8]

2.3.1 Senzory

Principy senzorů, které jsou součástí zařízení SensorTag CC2650, již byly popsány v kapitole 2.1. Zde je uveden souhrn a informace o údajích, poskytovaných zařízením SensorTag.

Tepelný senzor TMP007 – Měří teplotu objektu pomocí infračerveného světla ve stupních Celsia a teplotu okolí také ve stupních Celsia. Tento senzor se od června 2017 přestal v zařízeních SensorTag používat, ale je součástí zařízení, se kterým byla provedena implementace Droidsoru, a proto je zde zmíněn. Místo tohoto senzoru může být pro určení teploty použit senzor vlhkosti, anebo barometrický tlakový senzor.

Pohybový senzor MPU9250 – Zahrnuje gyroskop, měřící úhlovou rychlost v úhlech za sekundu, akcelerometr, měřící zrychlení gravitace G a magnetometr, který měří velikost a směr magnetické indukce v mikro Tesle.

Senzor vlhkosti HDC1000 – Měří relativní vlhkost (%rH) a teplotu ve stupních Celsia.

Barometrický tlakový sensor BMP280 – Měří tlak v hektopascalech (hPa) a teplotu ve stupních Celsia.

Optický senzor OPT3001 – Měří intenzitu světla v jednotkách Lux.

3 Platforma Android

Android je operační systém, založený na platformě Linux, určený pro mobilní telefony, tablety a rostoucí počet zařízení. Tento systém pohání více jak 2 miliardy zařízení. Aktuální verze je 8.1 s kódovým označením Oreo, avšak tato verze zatím není příliš rozšířená. Nejvíce rozšířené verze jsou v rozmezí od 4.4 až 7.1, jak jde vidět v tabulce číslo 1. Údaje v tabulce byly nashromážděny během 7denního období končícího 5. února 2018.

3.1 Android 4.4 - 8.1

V této podkapitole jsou probrány funkce, obsažené v jednotlivých verzích Androidu, které jsou podstatné pro implementovanou aplikaci.

Android 4.4 představil podporu pro hardwarové dávkování senzorů (anglicky "sensor batching"), což je optimalizace, která může radikálně snížit spotřebu u aplikací, pracujících se senzory. Pomocí tohoto dávkování pracuje Android s hardwarem zařízení, aby mohl přijmout a doručit data ze senzorů efektivně v dávkách, namísto individuálně ve chvíli, kdy jsou detekovány. Tato funkce je hardwarově závislá a nemusí se vyskytovat na všech zařízeních.

Android 5.0 přinesl hlavně grafické změny, a to nové grafické rozhraní Material Design a také nový vzhled notifikací.

Android 6.0 zavedl nový model správy oprávnění, kdy uživatelé mohou měnit oprávnění přímo za běhu aplikace, což poskytuje lepší přehled o tom, kdy je jaké oprávnění používáno. Z vývojářského hlediska to znamená, že aplikace musí zobrazovat požadavek o oprávnění až při běhu aplikace, namísto aby jej pouze uvedla v manifestu. Nutnost zobrazit požadavek se vztahuje pouze na nebezpečná oprávnění, jako například čtení obsahu z úložiště. Veškerá oprávnění musí i nadále být uvedena v manifestu.

Tato verze také představila optimalizace pro úsporu baterie anglicky nazývané "Doze" a "App Standby". Když je zařízení odpojeno od napájení, má vypnutou obrazovku a je v klidu, přejde do režimu Doze, který se snaží udržet systém v režimu spánku. V tomto režimu zařízení v krátkých časových úsecích periodicky pokračuje v normální činnosti, aby mohlo provést naplánované a opakující se operace. Režim App Standby umožňuje systému rozhodnout, že aplikace, která je v popředí, je v klidu, když ji uživatel nevyužívá a pokud je aplikace v klidu, vypne přístup k síti a pozastaví všechny operace, které aktivita prováděla.

Android 7.0 rozšiřuje režim Doze, kdy umožňuje do tohoto režimu přejít i když je zařízení v pohybu a aplikuje podobné opatření jako v Androidu verzi 6.0.

Tabulka 1: Zastoupení verzí Android [10]

Verze	Kódové označení	API	Zastoupení
2.3.3 - 2.3.7	Gingerbread	10	0,3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0,4%
4.1.x	Jelly Bean	16	1,7%
4.2.x	Jelly Bean	17	2,6%
4.3.x	Jelly Bean	18	0,7%
4.4	KitKat	19	12,0%
5.0	Lollipop	21	5,4%
5.1	Lollipop	22	19,2%
6.0	Marshmallow	23	28,1%
7.0	Nougat	24	22,3%
7.1	Nougat	25	6,2%
8.0	Oreo	26	0,8%
8.1	Oreo	27	0,3%

Android 8.0 omezil služby, běžící na pozadí, které například nemohou často získávat GPS pozici. Další změnu podstoupily notifikace, které musí být přiděleny určitému kanálu, například kanálu vytvořeném aplikací, nebo vůbec nebudou zobrazeny.

3.2 Bluetooth

Pro komunikaci s Bluetooth zařízeními na platformě Android je určeno API `android.bluetooth`, které podporuje jak Bluetooth Classic, tak od API verze 18 i Bluetooth Low Energy. [11]

3.3 Senzory

Třídy pro práci se senzory jsou součástí Android API `android.hardware`.

SensorManager – poskytuje přístup k senzorům.

Sensor – reprezentuje senzor.

SensorEvent – představuje událost senzoru a uchovává informace o senzoru, jako například čas pořízení záznamu, typ senzoru a jeho data.

SensorEventListener – slouží pro získání upozornění na nová data ze senzorů.

3.4 GPS

Pro získání GPS pozice slouží API `android.location`. V roce 2012 vydala společnost Google API balíček Google Play Services, který obsahuje API Fused Location Provider.

3.4.1 android.location

API podporované již od Android API verze 1. Umožňuje vybrat, jakým způsobem získat pozici (WiFi, nebo GPS), ale v dokumentaci je označeno za zastaralé a doporučuje se používat Fused Location Provider.

3.4.2 Fused Location Provider

Pro určení lokace bylo v API `android.location` nutno vybrat způsob, jakým se pozice získá. Při neoptimalizované práci s výběrem způsobu mohlo dojít ke zvýšené spotřebě baterie. Fused Location Provider se sám stará o volbu způsobu při optimální spotřebě baterie. Toto API je možné nastavit pro získání co nejpřesnější pozice na úkor spotřeby baterie nebo přibližné pozice s ohledem na spotřebu baterie. Fused Location Provider využívá některé třídy z API `android.location`.

3.5 Datové struktury platformy Android

Platforma Android poskytuje několik datových struktur, které se nacházejí v API `android.util`, které jsou navrženy s ohledem na úsporu pamětí. Datové struktury, se kterými pracuje i tato práce jsou mapy:

SparseArray – mapuje `int` na uvedený objekt.

SparseBooleanArray – mapuje celé číslo na `boolean` a `SparseIntArray`.

SparseIntArray – mapuje `int` na `int`.

Všechny zmíněné struktury fungují na stejném principu jako `HashMap`, ale oproti této struktuře šetří paměť, protože využívají primitivní proměnné jako klíče a nepoužívají objekty `Entry`. Dále nepřevádějí primitivní proměnné na jejich objektové ekvivalenty (například `int` na `Integer`) a díky tomu je méně potřeba garbage colletor. [12] Nevýhodou je pomalejší vyhledávání dat, kde namísto složitosti $O(1)$ je složitost $O(log\ n)$, ale při malém počtu položek je rychlost jak u `SparseArray` tak i u `HashMap` srovnatelná. [13] [14]

4 Vizualizace a záznam dat

Pro vizualizaci dat ze senzorů se nejčastěji využívají grafy, další možností je znázornit data trojrozměrně. Pro ukládání dat ze senzorů je nejvhodnější využít databázový systém, avšak občas může být vhodnější ukládat data jako čistý text nebo v nějakém datovém formátu, což usnadňuje přenos dat mezi zařízeními a systémy.

4.1 Grafy

Jak bylo zmíněno, k vizualizaci dat ze senzorů se nejčastěji používají grafy, kdy je možno využít jejich několik druhů.

Čárový graf – lze použít pro zobrazení průběhu měření jednotlivých senzorů v čase.

Sloupcový graf – při měření více senzorů lze tímto grafem vyjádřit počet dat z každého senzoru. Díky tomu lze vidět, který senzor měl největší zastoupení při měření.

Koláčový graf – může mít obdobné využití jako sloupcový graf.

Knihovny pro vykreslování grafů nejsou základní součástí Android API a proto je třeba vyhledat některou vhodnou. Za zmínku stojí například Androidplot[15], HelloCharts[16] a MPAndroidChart[17].

4.2 Trojrozměrné zobrazení

Méně častým způsobem pro vizualizaci dat je znázornit je v trojrozměrném prostoru, kdy lze na trojrozměrný obrazec, například krychli, aplikovat data ze senzoru a určit tak její natočení v prostoru.

Pro vykreslování trojrozměrných obrazců zahrnuje platforma Android podporu OpenGL, konkrétně OpenGL ES API. OpenGL je multiplatformní grafické API, které určuje standartní softwarové rozhraní pro grafický hardware, pracující s 3D grafikou. OpenGL ES je odnož specifikace OpenGL, zaměřená pro mobilní zařízení. Android podporuje několik verzí OpenGL ES. [18]

1.0 a 1.1 – Podporované od Androidem 1.0 a výše.

2.0 – Podporované od API verze 8 a výše.

3.0 – Podporované od API verze 18 a výše, je také třeba hardwarová podpora od výrobce zařízení.

3.1 – Podporované od API verze 21 a výše.

4.3 Záznam dat

Při záznamu dat je třeba tato data dlouhodobě uchovat. Nabízí se několik možností. Ukládat data do textového souboru v čistém textu, použít některý z datových formátů, anebo vkládat data do databáze.

4.3.1 Prostý text

Jedná se o nejzákladnější způsob ukládání dat. Velikost souboru je přímo úměrná množství textu, který se ukládá, nová data se jednoduše vkládají na konec souboru. Problém nastává při čtení a aktualizací takto uložených dat, kdy je nutné stanovit způsob, kterým budou data čtena a aktualizována a implementace těchto požadavků je znalostně i časově náročná.

4.3.2 Datové formáty

Datové formáty definují způsob, jakým se data do souboru zapisují a čtou, díky čemuž jsou tyto soubory snadno přenositelné a je možno je použít na libovolné platformě. Typickými příklady datových formátu jsou JSON a formáty založené na značkovacím jazyce XML. Úpravy těchto souborů mohou být u velkých souborů (nebo při častém zápisu) časově a paměťově náročné, a proto je v těchto případech lepší použít databázi. Avšak díky snadné přenositelnosti lze datové formáty použít pro export dat, uložených na zařízení v databázi. V případě implementované aplikace je třeba přenášet data ze senzorů spolu s GPS polohou, k čemuž lze použít například datový formát GPX.

4.3.3 Databáze

Databáze je kolekce informací, které jsou organizovány tak, aby byly snadno přístupné, spravované a aktualizované. Existuje více druhů databází, z nichž nejpoužívanější je relační databáze, která je tvořená z tabulek, obsahující data, která spadají do předdefinovaných kategorií. Platforma Android obsahuje relační databázi SQLite.

4.4 SQLite

SQLite je knihovna, implementující transakční SQL databázový engine, který je samostatný, k běhu nepotřebuje server a je snadno zprovoznitelný. Kód pro SQLite je volně dostupný na veřejné doméně a je možné jej použít pro jakýkoliv účel, ať už komerční nebo soukromý. SQLite je nejrozšířenější databáze na světě, kterou využívá i operační systém Android. SQLite provádí čtení a zápis přímo do běžných souborů na úložišti v zařízení. Celá databáze je uložena v jednom souboru, který je multiplatformní a je možné jej kopírovat mezi 32 a 64-bitovými systémy. [19]

4.5 GPX

GPX je datový formát XML, sloužící pro sdílení GPS údajů (trasových bodů, tras a cest) mezi aplikacemi a webovými službami na internetu. [20] Aktuální verze 1.1 byla vydána 4. dubna 2004, ale už při svém vydání v roce 2002 bylo GPX neoficiálním XML standardem pro sdílení GPS dat mezi aplikacemi.

Kromě samotných GPS informací umožňuje GPX formát uchovávat i jakékoliv další údaje, specifické pro aplikaci, která GPX soubor vytváří. V případě Droidsoru to jsou data ze senzorů.

5 Podobné aplikace

Hledání podobných aplikací bylo zaměřeno na to, aby svými funkcemi co nejvíce odpovídaly požadavkům implementované aplikace. Jednalo se o schopnost získávat data ze senzorů a GPS, provádět a ukládat záznam dat s možností následného zobrazení a exportu (nejlépe ve formě GPX), a možnost připojit se k externímu zařízení pomocí technologie Bluetooth a zobrazit jeho data. Nalezené aplikace vždy splňovaly jeden nebo více z uvedených požadavků.

5.1 Sensors Multitool

Aplikace je schopna získávat data ze senzorů a GPS pouze tehdy, když je aktivně používána. Data ze senzorů může ukládat pouze uživatel a pouze jednou za stisk tlačítka. Pro zobrazení více hodnot ze senzorů je použit čárový graf, avšak data z tohoto grafu není možné uložit. Při pokusu o export uložených jednotlivých záznamů se občas aplikace nečekaně ukončila. Uživatelské rozhraní je přehledné. Obsahuje vypnutelné reklamy.

5.2 Sensor Kinetics

Aplikace je velmi nepřehledná, což je způsobeno atypickým uživatelským rozhraním. Na druhou stranu obsahuje mnoho možností nastavení, například rychlost snímání senzorů, možnost snímat všechny senzory současně a také obsahuje názorné 3D ukázky, popisující funkce jednotlivých senzorů. Aplikace umožňuje export dat pouze v placené verzi.

5.3 Sensor Sense

Aplikace má uživatelsky přívětivé grafické rozhraní, ve kterém je snadné se zorientovat. Občas přestalo ukazovat hodnoty ze senzoru. Umožňuje nahrávat data senzorů a exportovat je do formátu csv. Během zobrazování a snímání dat je zobrazován čárový graf. Nesnímá při minimalizaci.

5.4 SenseView

Požaduje mnoho oprávnění. Zobrazuje notifikaci, upozorňující na běžící službu, kterou lze vypnout pouze v aplikaci při použití možnosti quit. Umí nahrávat záznam i v pozadí a tyto záznamy posléze exportovat do formátu zip, obsahujícího csv soubory každého senzoru. Neumí nahrávat současně interní a externí senzory. Z Bluetooth zařízení SensorTag umí získat pouze data z tepelného senzoru.

5.5 Android Sensors

Data ze senzorů jsou přehledně zobrazená na jedné obrazovce ve formě grafů. Zobrazuje také mapu, na které je znázorněna aktuální pozice zařízení. Provádí záznam všech senzorů a GPS pozice, při čemž zapíná službu, která je zobrazena v notifikačním poli a díky této službě snímá data i při minimalizované aplikaci. Data je možné zobrazit ve formě přehratelného záznamu, anebo exportovat v zip formátu, který obsahuje jednotlivé csv soubory.

5.6 nRF Connect

Aplikace pro obsluhu Bluetooth senzorů, ale nezahrnuje podporu interních senzorů Android zařízení. Je schopna zjistit všechny služby, nabízené senzorem, a zapnout dané služby, pokud uživatel zná postup. Data z externího senzoru zobrazuje bez předchozího zpracování. Zapíná službu na pozadí, na kterou neupozorňuje. Umí přijímat data i při běhu v pozadí.

5.7 SensorTag

Oficiální aplikace, určená ke komunikaci a zobrazení dat z externích senzorů typu SensorTag. V případě zařízení SensorTag CC2650 nesprávně zobrazuje data z barometru.

Aplikace byla dlouhou dobu neaktualizovaná a její starší verze využívala službu, na kterou neupozorňovala a která nikdy nebyla vypnuta, což způsobovalo zvýšenou spotřebu baterie. S poslední aktualizací byla tato chyba odstraněna.

5.8 Porovnání a vyhodnocení

Vyjma oficiální aplikace pro SensorTag umí každá z uvedených aplikací provádět záznam a následně jej exportovat. Pouze 3 aplikace umí provádět záznam i na pozadí a při vypnuté obrazovce. Z těchto 3 aplikací pouze jedna umí snímat jak interní, tak externí senzory, ale ani tato aplikace neumí provést export do formátu GPX. Tudíž žádná aplikace nesplňuje všechny požadavky, které si klade za cíl tato bakalářská práce.

Tabulka 2: Porovnání Android aplikací, které pracují se senzory

	Sensory		Záznam		
Jméno aplikace	Interní	SensorTag	Možnost záznamu	Na pozadí	Export
Sensors Multitool	Ano	Ne	Ano	Ne	Ano (txt)
Sensor Kinetics	Ano	Ne	Ano	Ne	Ano (placené)
Sensor Sense	Ano	Ne	Ano	Ne	Ano (csv)
SenseView	Ano	Ano	Ano	Ano	Ano (zip s csv)
Android Sensors	Ano	Ne	Ano	Ano	Ano (zip s csv)
nRF Connect	Ne	Ano	Ano	Ano	Ano (txt)
SensorTag	Ne	Ano	Ne	Ne	Ne
Droidsor	Ano	Ano	Ano	Ano	Ano (GPX)

6 Analýza a návrh

V této kapitole je probrán výběr verzí Android, které bude Droidsor podporovat, dále je ukázána analýza množství dat, které implementovaná aplikace vyprodukuje. Nakonec je proveden návrh služby, databáze a uživatelského rozhraní.

6.1 Výběr verze Androidu

Pro podporu technologie Bluetooth Low Energy je potřeba alespoň Android verze 4.3. Tuto verzi ale podle tabulky číslo 1 používá jen 0,7 % zařízení a do budoucna bude toto číslo ještě menší, proto není třeba tuto verzi podporovat a lze začít od verze 4.4. Vzhledem k tomu, že omezení operací v pozadí bylo uvedeno už v Androidu verze 6.0, tak po vyřešení tohoto omezení je možné podporovat i poslední vydanou verzi 8.1. O funkcích jednotlivých verzí, důležitých pro implementovanou aplikaci, se hovořilo v kapitole 3.1.

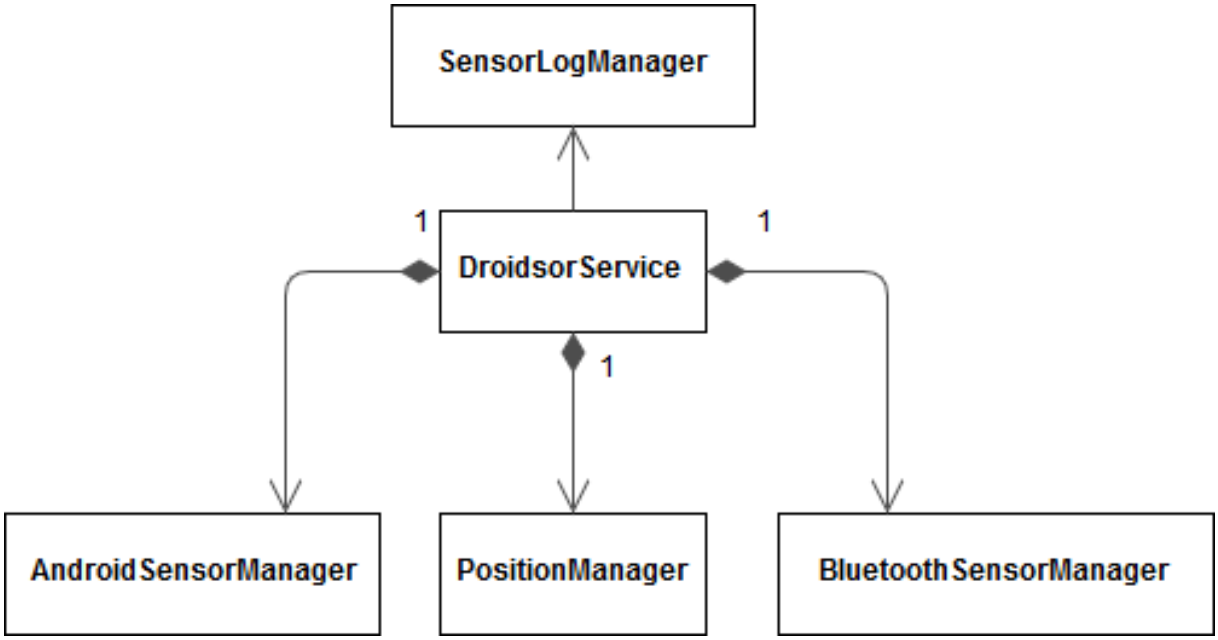
6.2 Odhadované množství dat

Největší objem dat bude v tabulce `Sensor_data`. Senzory, které implementovaná aplikace obsluhuje, poskytují maximálně 3 hodnoty (x, y, z) jako desetinná čísla. Dále je jako desetinné číslo zaznamenávána GPS pozice, konkrétně zeměpisná šířka a délka, výška, rychlost v metrech za sekundu a přesnost pozice GPS. Jako celé číslo jsou zaznamenávány id (primární klíč), statistická váha vzorku (použita pro rychlejší načítání záznamu), typ senzoru, který data poskytl a id záznamu, ke kterému daná položka patří.

Podle dokumentace SQLite dosahuje celé číslo velikost 1-8 bytů v závislosti na váze hodnoty a desetinné číslo má velikost 8 bytů. V tabulce je 9 sloupců s desetinnými čísly a 4 sloupce s celými čísly. Sloupce s typem senzoru a váhou vzorku zřídka překročí velikost 1 bytu. U ostatních sloupců se počítá s nejhorší variantou 8 bytů. Velikost jednoho záznamu je $8 \times 11 + 2 \times 1 = 90B$. Maximální interval snímání je 200 milisekund, což je 5 záznamů za sekundu a 18000 záznamů za hodinu. V nejhorším případě budou zaznamenávány všechny podporované senzory s maximální rychlostí snímání, což je 16 senzorů s intervalem záznamu 200 milisekund. Odhadované maximální množství dat v bytech za hodinu je $90 \times 18000 \times 16 = 25,92MB$.

6.3 Návrh služby

Služba je v případě Droidsoru srdcem aplikace. Zprostředkovává přenos dat mezi senzory a aplikací. Třídy `AndroidSensorManager` a `BluetoothSensorManager` se starají o obsluhu senzorů, jejich zapnutí, vypnutí a převod dat do přenositelné podoby. `SensorLogManager` zajišťuje správné spuštění a ukončení záznamu a zajišťuje, že součástí záznamu budou pouze údaje senzorů, obsažených v profilu. A nakonec třída `PositionManager` poskytuje nejaktuálnější zaznamenanou pozici z GPS.



Obrázek 2: Třídní diagram služby

6.4 Návrh databáze

V aplikaci je třeba dlouhodobě uchovávat profily a záznamy. Součástí profilů jsou položky, reprezentující senzory, které mají být součástí záznamu. Jeden senzor může být součástí více záznamů a jeden záznam může evidovat více senzorů, jedná se tedy o vztah M:N. V normálním případě by bylo potřeba vytvořit spojovou tabulku, ale v případě Droidsoru jsou informace o jednotlivých senzorech součástí aplikace a v databázi se vyskytuje pouze id senzoru. Proto jsou vytvořeny pouze dvě tabulky `Log_profiles` a `Log_profile_items`, kde `Log_profile_items` slouží jako spojová tabulka.

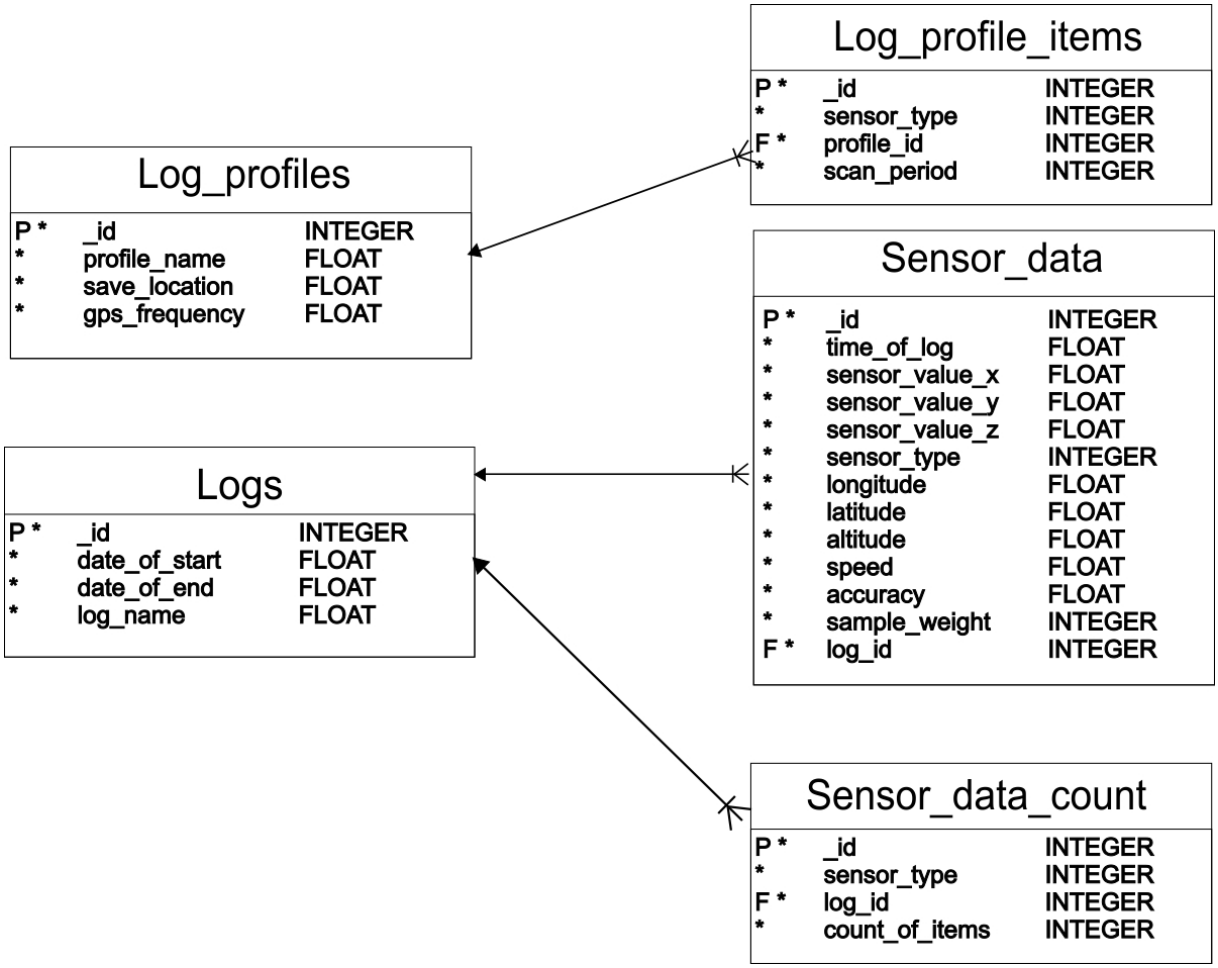
Součástí záznamů jsou položky, reprezentující data z jednotlivých senzorů. Záznam může mít více položek, ale položka může být součástí pouze jednoho záznamu. V tomto případě se jedná o vztah M:1 a vytvořené tabulky se jmenují `Logs` a `Sensor_data`. Pro zefektivnění načítání přehledu záznamu obsahuje tabulka `Sensor_data` sloupec, poskytující informaci o váze. Tato informace u každého senzoru označuje každou 10., 100. až 10000. položku v daném záznamu. Tato informace je využívána při načítání, kdy se podle počtu záznamů zvolí adekvátní váha. Pro omezení počtu dotazů ohledně počtu položek v daném záznamu je vytvořena tabulka `Sensor_data_count`, která údaje o těchto počtech uchovává.

Celkově se databáze skládá z pěti tabulek.

Log_profiles reprezentuje jeden profil záznamu.

Log_profile_items reprezentuje položky profilu záznamu.

Logs reprezentuje jeden záznam.



Obrázek 3: Relační model databáze

Sensor_data reprezentuje položku záznamu.

Sensor_data_count reprezentuje celkový počet výskytu senzoru daného typu v daném záznamu.

6.5 Návrh uživatelského rozhraní

Uživatelské rozhraní využívá Material Design, který se v Androidu využívá od verze 5.0, avšak je částečně dostupný i pro nižší verze díky knihovnám kompatibility.

6.5.1 Hlavní stránka

Hlavní stránka slouží pro zobrazení údajů ze všech aktivních senzorů. Je zde tlačítko, sloužící ke spuštění záznamu, jehož chování je nastavitelné. Tlačítko s ikonou Bluetooth umožňuje připojení k zařízení Bluetooth a pokud je zařízení již připojeno, umožňuje se od zařízení odpojit. Tato stránka také obsahuje boční menu (tzv. Drawer), díky kterému je možné zobrazit nastavení,

(a) Vzhled hlavní stránky (b) Vzhled správy profilu

Obrázek 4: Vzhled hlavní stránky a správy profilu

správu záznamových profilů, seznam nahraných záznamů a také umožňuje filtrovat zobrazované senzory.

6.5.2 Správa profilu

Zde je možné vytvářet nové profily a upravovat již exitující. Lze vybrat senzory, jejichž hodnoty se budou při záznamu snímat a interval snímání každého z nich. Také se zde nastavuje jméno profilu, určuje se, zda má profil snímat i GPS pozici a pokud ano, jak často.

6.5.3 Seznam nahraných záznamů

Zobrazuje seznam všech nahraných záznamů, které je možno jako celek přejmenovat, smazat anebo exportovat. Po kliknutí na záznam se zobrazí detail záznamu.

6.5.4 Detail záznamu

Detail záznamu pomocí grafu zobrazuje nahraná data ze senzorů. Data ze senzorů lze buď jednotlivě nebo hromadně exportovat ve formě GPX souboru. Grafové zobrazení je zde souhrnné a zobrazuje pouze omezený počet bodů, který lze změnit v nastavení. Po kliknutí na senzorovou položku se ukáže detailní zobrazení, obsahující všechny body.

6.6 Volba grafu

Jak bylo zmíněno v kapitole 4.1, platforma Android neposkytuje knihovny pro vykreslování grafu, a proto bylo třeba knihovnu vyhledat. Knihovna musela umět vykreslit údaje na více osách, označit tyto osy, umožnit vložit hodnoty času pro osu x a být schopna pracovat s více

Obrázek 5: Vzhled seznamu nahraných záznamů

jak 50000 body najednou. Z testovaných knihoven, které byly taktéž zmíněny v kapitole 4.1, splňovaly všechny knihovny většinu požadavků, ale rozhodujícím požadavkem bylo vykreslování velkého počtu bodů, kde Androidplot nebyl schopen tento počet v rozumném čase vykreslit a HelloCharts sice graf vykreslil, ale bylo téměř nemožné s grafem pracovat. Jediný MPAnroid-Chart zvládl velký počet bodů vykreslit a zpomalení se projevilo jen, když byl zobrazen celý graf najednou. Díky tomu, že splnila knihovna MPAndroidChart všechny požadavky, byla vybrána pro implementovanou aplikaci.

7 Implementace

Tato kapitola se zabývá implementací tříd a funkcí, které jsou nezbytné pro správný běh aplikace.

7.1 Služba

Jak bylo zmíněno v kapitole 6.3, služba je v případě Droidsoru srdcem aplikace a je zodpovědná za klíčové funkce jako začátek a ukončení záznamu i při vypnuté obrazovce, komunikaci se senzory a získávání polohy pro záznam. Aby bylo zajištěno, že služba nebude ukončena v rámci uvolnění místa v operační paměti a zároveň byla schopna pracovat a případně i získávat pozici GPS při vypnuté obrazovce, běží služba v popředí (anglicky je taková služba označována jako "Foreground Service"). Služba v popředí se typicky využívá u hudebních přehrávačů a kdykoliv (jako třeba záznam v případě Droidsoru), kdy provádí operaci, které si je uživatel vědom. Aby služba běžela v popředí, je třeba při startu služby v metodě `onStartCommand` zavolat metodu `startForeground`, kdy se jako argument funkce poskytuje objekt notifikace, která bude zobrazena v notifikační liště. Notifikace je u služeb v popředí povinná.

U API verzí, vyšších než 26, je navíc třeba služby v popředí spouštět pomocí metody `startForegroundService`, namísto metody `startService`. Voláním metody `startForegroundService` se spouštěná služba zavazuje, že do určité doby zavolá metodu `startForeground`. Pokud tak neučiní, bude služba zastavena.

V dalších několika kapitolách jsou popsány implementované třídy, které jsou odpovědné za zmíněné funkce služby.

7.2 Výčtový typ pro senzory

Pro jednodušší obsluhu interních i externích senzorů bylo třeba vymyslet způsob, jak senzory a jejich vlastnosti v aplikaci evidovat, aby byly lehce přístupné, poskytovaly požadovaná data a bylo snadné je v budoucnu rozšiřovat. Jelikož se vlastnosti senzorů při běhu aplikace nemění, je možné je považovat za konstanty, a proto byl implementován výčtový typ `SensorsEnum`, který všechny senzory zastřešuje a poskytuje jejich názvy, vlastnosti a metody pro převod dat do zobrazitelné podoby.

Protože každý senzor poskytuje jiný počet dat v jiných jednotkách, je třeba zavolat správnou položku výčtového typu pro získání relevantních dat. Toho by bylo možné dosáhnout tak, že by se prošly všechny položky výčtového typu a na základě porovnání poskytnutého id s id výčtového typu by se našla požadovaná položka. V nejhorším případě by se prošly všechny položky, což by při volání v cyklu mohlo způsobit zpomalení. Aby se tomuto problému předešlo, jsou při inicializaci výčtového typu všechny položky namapovány ke svým id pomocí objektu `SparseArray`, o kterém pojednávala kapitola 3.5.

Pro nalezení požadované položky se poté volá metoda `resolveSensor`, která jako parametr přijímá id sensoru a vrací nalezený výčtový typ, který už je schopen poskytnout požadované informace.

7.3 Interní senzory

Pro práci s interními senzory byla vytvořena třída `AndroidSensorManager` a tato kapitola popisuje klíčové funkce, které tato třída pro Droidsor poskytuje.

7.3.1 Výběr podporovaných senzorů

Platforma Android podporuje větší počet senzorů, než je běžně na jednom zařízení k dispozici a některá zařízení obsahují více senzorů stejného typu. Aby bylo možné zobrazovat správné údaje ze senzorů, je třeba určit, o jaký senzor se jedná, kolik údajů poskytuje a v jakých jednotkách. K tomu je využit již zmíněný `SensorsEnum`. Před použitím výčtového typu je třeba určit, které senzory jsou skutečně přítomné na daném zařízení. Toho lze docílit získáním seznamu všech senzorů pomocí metody `SensorManager.getSensorList(int)` s argumentem `Sensor.TYPE_ALL`. Po získání je třeba tento seznam projít a porovnat jeho položky se seznamem podporovaných senzorů a dojde-li ke shodě, zaznamenat přítomnost daného senzoru. V případě dvou senzorů stejného typu je třeba zajistit, aby se do seznamu přítomných senzorů nedostal dvakrát, čehož jde docílit pomocí mapy `SparseBooleanArray`, kdy se při prvním výskytu senzoru do této mapy vloží záznam a druhý senzor je díky existujícímu záznamu přeskočen.

Do Android API verze 3 byl podporován senzor orientace. Od vyšších verzí je třeba orientaci vypočítat a pro správný výpočet je potřeba získat data z akcelerometru a magnetometru. V Droidsoru je přítomnost senzoru orientace emulována, kdy jsou údaje o orientaci zobrazeny pouze tehdy, jsou-li zmíněné dva senzory přítomny na zařízení.

```
SparseBooleanArray foundSensors = new SparseBooleanArray();
//get all available sensors
List<Sensor> sensors= mSensorManager.getSensorList(Sensor.TYPE_ALL);
for(Sensor s: sensors){
   //If this type was not found yet remember it
   if(!foundSensors.get(s.getType(),false) && toListenIds.contains(s.getType())
       ){
      toListen.add(s);
      foundSensors.put(s.getType(),true);
   }
}
...
//If accelerometer and magnetometer are available remember also orientation
if(foundSensors.get(SensorsEnum.INTERNAL_ACCELEROMETER.sensorType,false) &&
```

```
foundSensors.get(SensorsEnum.INTERNAL_MAGNETOMETER.sensorType,false))
toListenIds.add(SensorsEnum.INTERNAL_ORIENTATION.sensorType);
```

Výpis 1: Výběr podporovaných senzorů

7.3.2 Získání dat ze senzorů

Pro získání údajů ze senzorů chytrého telefonu je použito rozhraní `SensorEventListener`, které poskytuje dvě metody, které je nutné implementovat. Jsou to metody `onSensorChanged(Sensor, int)` a `onAccurancyChanged(SensorEvent)`. Pro potřeby Droidsoru stačilo implementovat pouze `onSensorChanged(Sensor, int)`, neboť touto metodou jsou poskytovány nové údaje ze senzoru. Po implementaci zmíněné metody je možné získat aktuální data ze senzorů pomocí metody `registerListener(SensorEventListener, Sensor, int)`, která je součástí třídy `SensorManager` a jako parametry přijímá objekt, který má aktualizace zpracovávat, senzor, který má informace poskytovat a preferovanou četnost zasílání informací.

Získávat data ze senzorů lze pomocí libovolného objektu, který implementuje rozhraní `SensorEventListener` a má přístup k objektu `SensorManager`. Jelikož je třeba, aby při nahrávání záznamu bylo možné obdržet aktualizace ze senzorů i při vypnuté obrazovce nebo minimalizované aplikaci, je nejjednodušším řešením využívat třídu `AndroidSensorManager` až ve službě.

7.3.3 Frekvence senzorů

Podle dokumentace metoda `registerListener(SensorEventListener, Sensor, int)` umožňuje nastavit snímkovací frekvenci senzoru, ale tato frekvence je pouze orientační a události mohou nastat dříve i později, ale obvykle nastávají dříve. Jak bylo zmíněno v kapitole 3.1, existuje možnost získávat údaje ze senzorů se zpožděním ve formě dávky a zároveň snížit spotřebu baterie. Ale tato metoda pouze zpozdí odeslání dat, takže nakonec by bylo třeba příchozí data projít a vybrat jenom ta, která se vyskytla ve správném čase. Dalším negativem je fakt, že tento způsob není dostupný na všech zařízeních. Proto bylo třeba vytvořit systém pro správu frekvencí. Toho je docíleno použitím třídy `SparseIntArray`. Je potřeba dvou objektů této třídy. Jeden pro mapování požadované frekvence každého senzoru a druhý pro uložení času poslední aktualizace senzoru. Při obdržení nových dat v metodě `onSensorChanged` se z mapy evidující čas zjistí poslední aktualizace a vypočte se rozdíl se současným časem. Je-li rozdíl větší než frekvence získána z druhé mapy, je nový údaj zpracován a údaje v mapě aktualizovány.

7.4 Komunikace s externím senzorem

Pro komunikaci s externím senzorem je potřeba, aby chytrý telefon podporoval technologii Bluetooth Low Energy. Pro komunikaci s externím senzorem byla vytvořena třída `BluetoothSensorManager`, která je zodpovědná za všechny operace popsané v této kapitole.

7.4.1 Vyhledání zařízení

Vyhledání zařízení se provádí za pomocí třídy `BluetoothAdapter`, která obsahuje metody `startLeScan` a `stopLeScan`. Tyto metody přijímají objekt `BluetoothAdapter.LeScanCallback`, který implementuje metodu `onLeScan`. Tato metoda bude zavolána pokaždé, když je nalezeno nové Bluetooth zařízení.

Po zobrazení všech dostupných zařízení musí uživatel zvolit SensorTag. Po zvolení zařízení proběhne pokus o spojení.

7.4.2 Spojení s zařízením

Pro správu spojení se zařízením se používá třída `BluetoothGatt`. Pokud se jedná o první připojení, je třeba získat referenci pomocí metody `BluetoothDevice.connectGatt`[1], kde třída `BluetoothDevice` reprezentuje BLE zařízení. Metoda `connectGatt` přijímá jako parametr objekt `BluetoothGattCallback`, který informuje o změnách, které nastaly v BLE zařízení, jako například změna stavu spojení, objevení nových služeb a informace o dokončení čtení nebo zápisu do zařízení. Metody implementující toto chování jsou popsány v kapitole 7.4.3.

```
//Reconnect to device
if(mBluetoothDeviceAddress != null && address.equals(mBluetoothDeviceAddress)
    && mBluetoothGatt !=null){
   if(mBluetoothGatt.connect()){
      mConnectionState = STATE_CONNECTING;
      return true;
   } else return false;
}
//Connect to found device
final BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
if(device == null)return false;
mBluetoothGatt = device.connectGatt(sensorService,false,myBluetoothGattCallback
    );
```

Výpis 2: Připojení k Bluetooth zařízení

Po navázání spojení se zařízením se spustí metoda `BluetoothGattCallback.onConnectionStateChanged`, která indikuje stav spojení a s níž je možné začít hledat služby, nabízené BLE zařízením.

[1] Pokud již došlo k připojení k zařízení a spojení bylo zrušeno, reference na objekt zůstává a je možné se znovu připojit k zařízení pomocí metody connect.

7.4.3 Zapnutí senzorů

V metodě `onConnectionStateChanged`, je možné začít hledat dostupné služby na připojeném zařízení pomocí metody `BluetoothGatt.discoverServices`. Určité služby poskytují prostředky pro práci se senzory (zapnutí, vypnutí, nastavení frekvence). Při objevení služeb, se spustí metoda `BluetoothGatt.onServicesDiscovered`, kde je třeba dané služby projít a najít ty, které slouží pro práci se senzory. Služby jsou identifikovány pomocí UUID a díky tomuto identifikátoru je možné je jednoznačně přiřadit k danému senzoru. Každá služba obsahuje charakteristiky, což jsou zmíněné prostředky pro správu senzoru. Stejně jako služba samotná, tak i charakteristiky jsou identifikovány pomocí UUID. Po získání reference na všechny požadované služby a charakteristiky je možné začít senzory postupně zapínat.

```
for (BluetoothGattService s : mBluetoothGatt.getServices()){
   //Iterate list of searched sensors and find required services
   for(GeneralTISensor sensor: sensors){
      if(sensor.resolveService(s)){
         //Service found
         activeSensors.add(sensor);
         areActiveSenorsSet = true;
         break;
      }
   }
}
// start found sensors
initializeSensors();
```

Výpis 3: Získání služeb

Komunikace mezi BLE zařízením a chytrým telefonem je založena na zprávě a odpovědi. Po odeslání zprávy je nutno počkat na odpověď, a až poté je možné odeslat další zprávu. Pro čekání na odpověď je využita třída `Semaphore`, kdy se po každém odeslání čeká na povolení pokračovat. Odpověď se v případě zápisu do sensoru vrací zpátky pomocí metod `onCharacteristicWrite` nebo `onDescriptorWrite` v závislosti na objektu, který se zapisoval, a právě v těchto metodách je uděleno povolení pokračovat. Aby bylo možné použít třídu `Semaphore`, je metoda pro zapnutí senzorů spouštěna v samostatném vlákně.

Než se zapne senzor, je třeba zapnout notifikaci na daném senzoru, čímž se zajistí, že senzor bude periodicky posílat data. Notifikace se nastavuje pomocí deskriptoru, jenž je součástí charakteristiky, starající se o data a má v případě SensorTagu vždy stejné UUID. Po zapnutí notifikace je možné zapnout samotný senzor a také mu nastavit frekvenci. Po nastavení notifikace a zapnutí senzoru se nová data ze senzoru objeví v metodě `onCharacteristicChange`.

```
for(GeneralTISensor sensor: sensors){
   //Determines wheter the sensor should be turned on or off
   boolean enable = activeSensors.contains(sensor);
   sensor.configureNotifications(enable);
   //Wait for communicationSemaphore.release() method call
   communicationSemaphore.acquire();
   sensor.configureSensor(enable);
   communicationSemaphore.acquire();
   if (enable){
      //Configure frequency if this sensor will be used
      sensor.configureSensorFrequency(listenFrequencies.get(sensor.
         getSensorType(), 1000));
      communicationSemaphore.acquire();
   }
}
```

Výpis 4: Zapnutí senzorů

Používaná třída `GeneralTISensor` je abstraktní, poskytující základní metody pro obsluhu jednotlivých senzorů. Jednotlivé typy senzorů z této třídy dědí a přepisují metody u těch senzorů, které vyžadují jiný postup k jejich zprovoznění.

```
//Turns on or off the sensor
public void configureSensor(boolean enable){
   configurationCharacteristic.setValue(enable?new byte[]{0x01}:new byte[]{0x00
      });
   mBluetoothGatt.writeCharacteristic(configurationCharacteristic);
}
public void configureNotifications(boolean enable){
   BluetoothGattDescriptor desc = dataCharacteristic.getDescriptor(
      CLIENT_CHARACTERISTIC_CONFIG);
   mBluetoothGatt.setCharacteristicNotification(desc.getCharacteristic(),enable
      );
   desc.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
   mBluetoothGatt.writeDescriptor(desc);
}
public void configureSensorFrequency(int frequency){
   //Ensure that frequency is in supported bounds
   if (frequency > 2450) frequency = 2450;
   if (frequency < 100) frequency = 100;
   //Preccision is 10ms
```

```
    byte f = (byte)((frequency/10));
    periodCharacteristic.setValue(new byte[]{f});
    mBluetoothGatt.writeCharacteristic(periodCharacteristic);
}
```

Výpis 5: Metody pro obsluhu senzoru

7.4.4 Interpretace dat ze senzoru

Po zavolání metody `onCharacteristicChange` je třeba zjistit, jaký senzor data poskytl pomocí UUID. Po nalezení odpovídajícího senzoru se musí data, obsažená v charakteristice, převést podle dokumentace do čitelné podoby. V případě Droidsoru byla pro převod použita již existující třída, která je součástí oficiální aplikace pro SensorTag, ale tato třída obsahovala chybu při převodu údajů z barometru. Chyba spočívala v přístupu ke špatnému indexu v poli, po nastavení správného indexu převod fungoval korektně.

7.5 Získávání polohy

U získávání polohy je důležité, aby bylo možno získat polohu ze služby, jinak by nebylo možné při nahrávání záznamu získávat polohu při minimalizované aplikaci. Samotné získávání polohy ze služby možné je, avšak pro získání polohy je třeba, aby bylo GPS zapnuto a aplikace měla oprávnění získávat polohu a pro korektní zajištění těchto požadavků je potřeba aktivita. Proto je třída `PositionManager`, který byla napsaná pro Droidsor, aby pomáhala získávat polohu, schopna pracovat ve dvou režimech. V režimu, kdy je aktivita dostupná a v režimu práce ze služby. Třída režimy nerozlišuje a chování je určeno podle zavolané metody.

Režim dostupné aktivity se používá těsně před spuštěním záznamu, kdy se provede pokusné získání polohy, při kterém se provede ověření oprávnění a zjistí se, je-li zapnutý GPS přijímač. Pokud ne, je o tom uživatel informován a vyzván k povolení zakázané položky. V opačném případě se spustí samotný záznam.

Při spuštění záznamu se ve službě zavolá metoda objektu `PositionManager`, která je v režimu práce ze služby a slouží čistě pro obdržení GPS pozice, kdy počítá s tím, že veškerá oprávnění již byla udělena a pokud ne, žádnou pozici nezískává. Za předpokladu, že jsou všechna oprávnění udělena a GPS je zapnuta, provádí tento režim také předběžné získání pozice při prvním zapnutí služby. Úspěšné předběžné získání pozice zajišťuje, že záznam, co nahrává i pozici, bude mít už od první položky dostupnou alespoň přibližnou GPS pozici.

Poloha není získána okamžitě, ale data ze senzorů mohou být uložena několikrát za sekundu, a proto se poslední pozice ukládá do proměnné, která se poskytuje všem datům ze senzorů. Po získání nové polohy se proměnná aktualizuje, aby poskytovaná pozice byla co nejpřesnější.

7.6 Content Provider

Content Provider pomáhá aplikaci spravovat přístup k datům, spravovaným danou aplikací nebo uloženým v jiné aplikaci a umožňuje je sdílet s ostatními aplikacemi. Co se týče přístupu k datům jiných aplikací a sdílení dat, Droidsor tyto funkcionality nevyužívá a pro přístup k datům jde použít přímočařejší třídu `SQLiteOpenHelper`. Ale služeb Content Provideru využívají třídy `CursorAdapter` a `CursorLoader`, které jsou v Droidsoru používány. Další výhodou Content Provideru je, že se sám stará o otevírání a ukončování spojení s databází, což je užitečné při práci s více vlákny, anebo v případě Droidsoru přístupu k databázi ze dvou míst (Služba a aktivita). Proto Droidsor využívá právě Content Provider.

7.6.1 Vytvoření Content Provideru

Aby bylo možné Content Provider používat je třeba vytvořit třídu, která dědí ze třídy `Content-Provider`. Implementace nově vzniklé třídy závisí na druhu zdroje dat. V případě Droidsoru je zdrojem dat SQLite databáze.

Content Provider pracuje s URI adresami, kdy podle URI adresy, poskytnuté při volání Content Provideru, rozhodne, na kterou tabulku se má provést dotaz. Adresy od sebe rozlišuje pomocí třídy `UriMatcher`.

```
//Base path to this provider
private static final String AUTHORITY = "com.example.marekulip.provider";
private static final UriMatcher sUriMatcher = new UriMatcher(UriMatcher.
    NO_MATCH);
//address with which is this Content Provider accessed
public static final Uri LOG_PROFILE_URI = Uri.parse("content://"+ AUTHORITY+"/"
    +LogProfilesTable.TABLE_NAME);
static { //Add address to UriMatcher
   //Adress for whole table
   sUriMatcher.addURI(AUTHORITY, LogProfilesTable.TABLE_NAME,1);
   //Address for one row in a table
   sUriMatcher.addURI(AUTHORITY, LogProfilesTable.TABLE_NAME+"/#",2);
}
```

Výpis 6: Vytvoření URI adres

Po vytvoření URI adres je třeba implementovat metodu `onCreate`, která se stará o inicializaci zdroje dat. Nakonec je nutno implementovat všechny CRUD metody.

```
switch (sUriMatcher.match(uri)){
   case LOG_PROFILE: // Same as query SELECT * FROM Table
```

```
        queryBuilder.setTables(LogProfilesTable.TABLE_NAME);
        break;
    // Query would look like SELECT * FROM Table WHERE _ID = ?
    case LOG_PROFILE_ID:
        //Set the table which will be queried
        queryBuilder.setTables(LogProfilesTable.TABLE_NAME);
        //Add argument WHERE _ID = ?
        queryBuilder.appendWhere(LogProfilesTable._ID + " = " + uri.
            getLastPathSegment());
        break;
}
SQLiteDatabase db = database.getReadableDatabase();
Cursor cursor = queryBuilder.query(db,projection, selection, selectionArgs,
    null, null, sortOrder);
cursor.setNotificationUri(getContext().getContentResolver(),uri);
return cursor;
```

Výpis 7: Implementování CRUD metody query

Po implementování CRUD metod je možné provádět volání na daný Content Provider.

7.6.2 Přístup k datům

Pro získání reference na Content Provider se volá metoda `getContentProvider` ze třídy `Context`. Objektů třídy `Context` existuje více druhů, dva důležité jsou kontext, který má každá aktivita a aplikační kontext, ke kterému lze přistoupit metodou `Context.getApplicationContext`. Dokumentace doporučuje při používání třídy `ContentProvider` používat aplikační kontext, který není závislý na životním cyklu `Activity`. Při používání kontextu aktivity může docházet k výjimce `NullPointerException`, když se dotaz nestihne dokončit před ukončením aktivity.

Po získání reference se volá metoda v závislosti na tom, jaká operace je požadována.

```
//SELECT * FROM Log_profile WHERE log_id = ?
getApplicationContext().getContentResolver().query(DroidsorProvider.
    LOG_PROFILE_URI,
null,
SensorDataTable.LOG_ID+ " = ?",
new String[]{String.valueOf(id)},
null);
```

Výpis 8: Volání query metody Content Provideru

7.7 Snímání dat

Nahrávání záznamu je realizováno pomocí služby Droidsoru, která má odkaz na objekt `SensorLogManager` (dále označován jako správce záznamu), jenž se stará o ukládání záznamů. Služba slouží jako prostředník, kdy přijímá data od senzorových správců a přeposílá je správci záznamu, pokud probíhá záznam. Před samotným spuštěním záznamu získá správce záznamu oprávnění z objektu `WakeLock`, který zajistí, aby nebyl procesor v průběhu záznamu vypnut. Následně správce záznamu ověří, zda-li data mají být součástí snímaného záznamu a pokud ano, uloží je do dočasného pole. Obsah dočasného pole se každých deset sekund ukládá do databáze pomocí takzvaného bulk insertu. O opakované provedení tohoto úkonu se stará třída `Timer`. Po uložení do databáze se obsah dočasného pole vymaže, aby položky pole zbytečně nespotřebovávaly paměť.

Při ukončení záznamu dojde k uložení zbývajících dat, ukončí se časovač a senzory se vrátí do původního stavu.

Touto kapitolou byly popsány všechny třídy, odpovědné za správné fungování služby. Další kapitoly se zabývají hlavními problémy, které nastaly při implementaci a jak byly řešeny.

7.8 Zobrazení aktuálních dat

Aby bylo možné poslat data ze služby do objektu `Activity`, je nutné zavolat metodu `sendBroadcast`, která jako parametr přijímá objekt typu `Intent`, jenž obsahuje informace o daném vysílání a obsahuje jeho data. Senzory jsou schopny získat data několikrát za sekundu a každá nově odeslaná data do aktivity způsobí, že `ListView`, pomocí kterého aktivita data zobrazuje, se celý vykreslí znovu. Pokud by tedy posílání nových dat nebylo nějak regulováno, přestala by aplikace odpovídat, neboť by byla zcela vytížena zpracováváním jednotlivých zpráv ze služby. Řešením je po určitou dobu (například minimální možnou frekvenci, která jde v aplikaci nastavit) data získávat, ale odeslat je až po uplynutí stanovené doby. S tímto řešením se ale ve službě hromadí data, která už nejsou aktuální a nebudou použita. Proto jsou data, získávaná během doby čekání, ukládána do mapy `SparseArray` a současně se do seznamu `ArrayList` ukládají id všech senzorů, která se po dobu čekání vyskytla. Dalším problémem je, že údaje ze senzorů jsou umístěny v objektu a ten bez serializace nelze umístit do objektu `Intent`. Řešením je při zavolání metody `sendBroadcast` neposlat data, ale pouze indikátor, že nová data jsou připravena k vyzvednutí. Po obdržení tohoto indikátoru zavolá aktivita metodu služby, díky které zjistí, došlo-li vůbec k nějaké změně, a pokud ano (seznam má hodnotu `null`), získá se ze služby i mapa, obsahující data senzorů, která jsou následně zobrazena pomocí zmíněného `ListView`.

7.9 Zobrazení souhrnu záznamu

Zobrazení souhrnu záznamu zahrnuje tři úkony. Načtení dat záznamu z databáze, převedení těchto dat do manipulovatelné podoby a vložení dat do grafu, aby mohla být zobrazena. Nejnáročnější operací je převedení dat do manipulovatelné podoby. Tato operace je závislá na množství dat, která jsou načtená, a proto je důležité zaměřit se především na jejich objem. V souhrnu

nemusí být načtena všechna data, ale jenom počet dostatečný pro znázornění průběhu záznamu. Nabízí se tři možnosti.

První možnost pro načtení dat je neřešit množství dat a načíst veškerá nahraná data pro daný záznam. Toto řešení je ideální pro malý objem dat, protože se provede pouze jeden dotaz a převedení dat není časově náročné, ale v případě velkých záznamů způsobuje převedení dat do manipulovatelné podoby dlouhé čekací doby.

Druhá možnost je si při záznamu evidovat, kolik položek daného senzoru bylo pořízeno a při určitém počtu udělat u daného záznamu značku. Například váhu, která označí každý 10., 100. až 10000. záznam. Vyšší váhy nedávají smysl, protože by dříve došla paměť SD karty, než by je bylo možné použít. Poté se při načítání nejdříve zjistí, kolik záznamů daného senzoru bylo uloženo, zvolí se váha pro každý senzor podle počtu a provede se dotaz do databáze pro každý senzor na základě váhy. Bohužel nelze zvolit jednotnou váhu, protože některé senzory nemusely poskytnout dostatek dat, aby se v dané váze jejich výsledky projevily. Při tomto řešení se značně urychlí převedení dat, ale zvýší se počet dotazů, což při větší velikosti databáze způsobí značné zpomalení ve fázi načítaní dat. Tento problém se nejvíce projeví u záznamu s velkým počtem snímaných senzorů.

Třetí možnost je spojit výhody první a druhé možnosti – použít váhy a jenom jeden dotaz, který bude v klauzuli `WHERE` obsahovat všechny podmínky pro všechny senzory a váhy. Tímto způsobem je dosaženo kompromisu, kdy data jsou načtena v přijatelném čase a převedení dat je provedeno díky malému počtu dat téměř okamžitě.

Při použití vah může nastat problém v případě nekorektního ukončení záznamu, kdy se nezapíšou výsledné počty položek pro každý senzor. Bez těchto údajů není možné souhrny zobrazit, protože aplikace nejdříve získá počty a z nich odvodí senzory, na které se má dotazovat. Řešení tohoto problému je vytvořit při začátku záznamu umělé počty položek a při ukončení záznamu tyto počty aktualizovat o výsledné hodnoty. Pokud dojde k nekorektnímu ukončení záznamu, bude souhrn zobrazitelný, ale bez použití vah.

Nevýhodou vah je, že nelze vybrat přesný počet bodů, které budou zobrazeny a vždy lze zajistit jen určitý rozsah, a tak může nastat případ, že senzor vygeneroval 1200 položek, a je-li rozsah nastaven na 100-1000 položek, zobrazí se pouze 200 položek, zatímco u senzoru, který vygeneroval 900 položek by byly při zmíněném rozsahu zobrazeny všechny položky. Rozsah může uživatel měnit a podle jeho aktuálního nastavení se volí váha. Volba váhy probíhá tak, že se v cyklu hledá váha, která by splnila požadovaný rozsah a zároveň zobrazila co nejvíce položek v rámci jeho nastavení. Váhy se procházejí od té s největším rozptylem (vrací nejméně položek) a pokud počet položek, které by zvolená váha poskytla překročí požadovaný rozsah, zvolí se poslední platný počet položek. Pokud neexistuje platný počet položek, zvolí se první váha, která překročila požadovaný rozsah (tato váha je nejblíže požadovanému rozsahu).

```
if(count<prefferedCount)return 1;
boolean hasValidIndex = false;
int validIndex = 0;
```

```
for(int i = 0, countedCount; i < SensorLog.weights.length; i++){
   countedCount = count/SensorLog.weights[i];
   if(countedCount >= 1 && countedCount < prefferedCount){
      hasValidIndex = true;
      validIndex = i;
      continue;
   }
   if(hasValidIndex) return SensorLog.weights[validIndex];
   if(countedCount > prefferedCount) return SensorLog.weights[i];
}
return 1;
```

Výpis 9: Volba váhy

7.10 Změna orientace obrazovky

Za určitých okolností může nastat případ, kdy změna orientace obrazovky způsobí pád aplikace. Nejvíce nebezpečné jsou případy, kdy probíhá dlouhá operace, je zobrazen dialog, který počítá s objektem v aktivitě, anebo dochází k práci se službou.

7.10.1 Dlouhé operace

Pokud dojde ke změně orientace při vykonávání dlouhé operace a operace byla vykonávána ve fragmentu, může se stát, že volání funkce `getActivity` vrátí `null` i přesto, že aktivita stále existuje. Proto je potřeba při dlouhých operacích ověřit, zda-li je reference na aktivitu platná. Pokud není reference platná, jsou data zahozena a budou znovu načtena při připojení fragmentu k aktivitě.

7.10.2 Práce se službou

Při změně orientace dochází ke ztrátě spojení se službou. Pokud došlo ke změně orientace v aktivitě, kde se pracuje se službou, nedochází k žádným potížím, protože životní cyklus aktivity zajistí, aby bylo spojení se službou obnoveno. Problém nastává v případě, kdy byla otevřena nová aktivita metodou `startAcitivityForResult`, a v této aktivitě došlo ke změně orientace. Výsledek aktivity se vrací v metodě `onAcitivityResult`, ale tato metoda je zavolána dříve než metody `onResume` nebo `onCreate` a při změně orientace dochází ke ztrátě spojení, které je obnoveno až v metodě `onResume`. Důsledkem toho neexistuje při návratu výsledku reference na službu a nebude obnovena dříve, než se zpracuje výsledek, ale zpracování výsledku je v některých případech závislé na přítomnosti služby. Pokud dojde v takovém případě k volání služby, nastane výjimka `NullPointerException`. Řešením je ověřit, zda-li je platná reference na službu a pokud ne, vytvořit nové vlákno, ve kterém se uchová výsledek, dokud nebude spojení

se službou obnoveno. Čekání je realizováno pomocí třídy `Semaphore`, která funguje na základě získání a uvolnění povolení. V nově vytvořeném vlákně požádá semafor o povolení, které získá po obnovení spojení se službou. Po udělení povolení je možné opět volat metody ze služby a dokončit zpracování výsledku aktivity.

7.10.3 Dialogy

Pokud dialogové okno není při změně orientace zrušeno a jeho výsledky spoléhají na objekty z aktivity, může se stát, že dialog zavolá po změně orientace metodu, která už nemá odkaz na objekt, se kterým se počítalo. Aby se tomuto zamezilo, poskytuje třída `Activity` metody, které jsou volány při restartu aktivity a je to metoda sloužící pro uložení stavu aktivity `onSaveInstanceState` a metoda pro obnovení stavu aktivity `onRestoreInstanceState`. Uložit lze celé objekty pomocí serializace, ale tyto operace mohou být časově náročné. Namísto ukládání celých objektů je vhodnější uložit si pouze prostředky pro obnovení těchto objektů, jako například v případě Droidsoru id profilu, se kterým má probíhat záznam. Po obnovení stavu je profil načten pomocí uloženého id a je opět možno bezpečně volat z dialogu metody.

8 Testování

Při testování byla největší pozornost zaměřena na spotřebu baterie a paměti. Dalšími oblastmi, na které se testování zaměřilo, bylo chování aplikace při použití třídy `WakeLock` a bez ní, rychlost zobrazení souhrnu záznamu pomocí různých způsobů, a nakonec rychlost připojení a odpojení s externím senzorem.

8.1 Spotřeba baterie

Největší vliv na spotřebu baterie má aplikace ve chvíli, kdy probíhá záznam. Pokud služba běží, aplikace je minimalizována nebo vypnutá a neprobíhá záznam, je spotřeba aplikace nulová. Při zobrazování údajů ze senzorů na hlavní obrazovce Drodidsoru je spotřeba srovnatelná se spotřebou při zapnuté obrazovce.

Při probíhajícím záznamu se již spotřeba projeví na výdrži baterie. Pokud záznam snímá senzory, ale neukládá GPS pozici, je spotřeba srovnatelná se stavem, kdy je zakázán spánek. Při získávání pozice už dochází k výraznému úbytku výdrže baterie, kdy je úbytek závislý na četnosti vyhodnocování GPS pozice. Pokud je pozice získávána maximální rychlostí, může být spotřeba i 10 % – 12 % za hodinu. Při snížení frekvence na 1 pozici za minutu se spotřeba za hodinu pohybuje kolem 5 %.

8.2 Záznam bez zakázání spánku

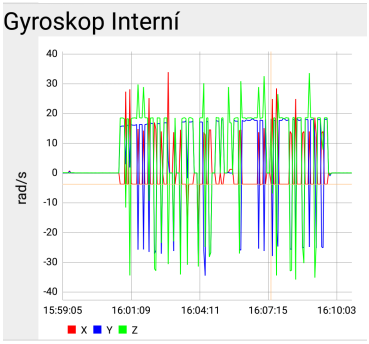
Pokud aplikace před samotným záznamem nezakáže spánek, může zajistit korektní data jen tehdy, když záznam probíhá při zapnuté obrazovce. V opačném případě dochází k nečekaným výsledkům, kdy v horším případě záznam probíhá dál, ale zapisuje nekorektní hodnoty a v lepším případě neprobíhá zápis dat. Tato chyba se nemusí projevit okamžitě, získává-li záznam i GPS pozici, kdy získávání GPS pozice spánek zakazuje, ale i v tomto případě může ke zmíněným chybám dojít, i když nejsou tak časté.

8.3 Objem nasnímaných dat

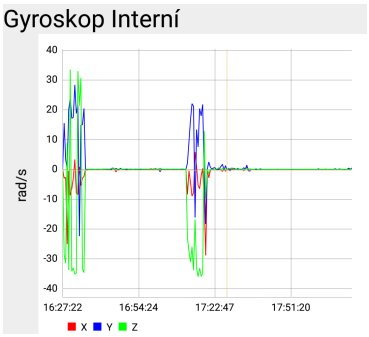
Při testování byl použit profil, který snímal všechny dostupné interní senzory a všechny součásti externího senzoru, což ve výsledku bylo 13 senzorů, snímaných rychlostí 200 milisekund včetně GPS pozice. Výsledná velikost záznamu byla 12,72 MB, což je více jak o polovinu méně oproti odhadované velikosti. Tento rozdíl je způsoben tím, že nebyl snímán plný počet senzorů. Některé senzory poskytují data pouze při změně, a to nikoliv tak často, jak je požadováno.

8.4 Rychlost načtení souhrnu záznamu

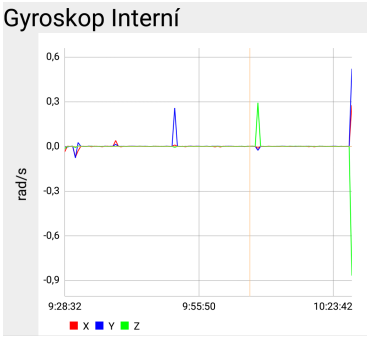
Jak je zmíněno v kapitole 7.9, byly při implementaci zvažovány tři způsoby implementace načtení souhrnu – načtení všeho, načtení pomocí vah s více dotazy nebo načtení pomocí vah s jedním dotazem. Při testování byly tyto způsoby porovnány v rámci rychlosti. Ve většině případů byl



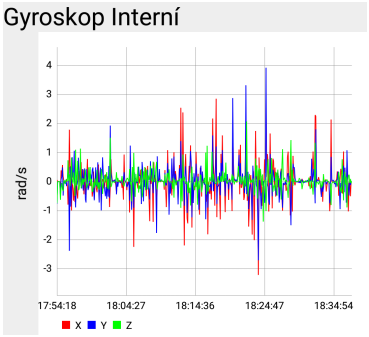
(a) Bez zámku a GPS v klidu



(b) Bez zámku s GPS v klidu



(c) Se zámkem bez GPS v klidu



(d) Se zámkem v pohybu

Obrázek 6: Porovnání záznamů vzhledem k zakázání spánku

nejrychlejší třetí způsob, který velké množství dat načetl nejrychleji a malé objemy dat zobrazil téměř stejně rychle, jako první. U zbylých dvou způsobů záleží na počtu načítaných senzorů a délce záznamů, kdy při malém počtu senzorů a dlouhé době záznamu byl rychlejší druhý způsob a naopak při velkém počtu senzorů a krátké době záznamu zvítězil první způsob. Naměřené hodnoty jsou uvedeny v tabulce 3. Do naměřeného času je zahrnuto získání dat z databáze, jejich zpracování a vložení do grafu. Způsoby byly volány po sobě v pořadí, jaké je v tabulce. Druhé a třetí volání je nepatrně rychlejší, než kdyby byly volány samostatně.

Tabulka 3: Porovnání rychlosti načtení souhrnu záznamu

			S váhami	
Délka záznamu	Počet senzorů	Jeden dotaz(vše)	Více dotazů	Jeden dotaz
00:05	1	712 ms	651 ms	618 ms
00:08	6	765 ms	7694 ms	676 ms
01:07	6	1034 ms	4264 ms	964 ms
19:00	6	17156 ms	4547 ms	1369 ms
29:46	13	38640 ms	11198 ms	3582 ms

8.5 Rychlost připojení k Bluetooth zařízení

Jak je zmíněno v kapitole 2.2, připojení k zařízení, využívající technologii BLE, by mělo být téměř okamžité. Avšak při testování k okamžitému připojení nedocházelo pokaždé, někdy připojení trvalo i půl minuty. Tento problém byl výraznější u Androidu verze 7.1, kdy i odpojení trvalo neobvykle dlouhou dobu.

8.6 Externí senzor se slabou baterií

Pokud má externí senzor slabou baterii, poskytuje nepřesná, nesmyslná, anebo vůbec žádná data. U pohybového senzoru trvalo více než minutu, než začal poskytovat data ze všech jeho částí a senzor vlhkosti občas vykazoval vlhkost přesahující 100 %. Po výměně baterie se tyto problémy nadále nevyskytovaly.

9 Závěr

V úvodu této práce byly probrány senzory a jejich fyzikální principy, komunikace s externím senzorem a popis technologií, které komunikaci zprostředkují, byly probrány prostředky platformy Android, kterých implementovaná aplikace využívá a byly uvedeny možnosti pro vizualizaci a záznam dat. Dále byla provedena analýza podobných aplikací, která dokázala, že doposud neexistuje aplikace se stejnými požadavky, jako implementovaná aplikace Droidsor.

Následně byla provedena analýza a návrh aplikace, podle které byla vytvořena. Výsledná aplikace Droidsor dokáže zobrazit data z devíti interních senzorů a je schopna přečíst údaje z externího senzoru SensorTag CC2650. Umí tato data dlouhodobě nahrávat, ukládat spolu s GPS pozicí, zobrazovat ve formě grafu a také exportovat ve formě GPX.

Nakonec byla implementovaná aplikace otestována na třech zařízeních a následně publikována v online distribuční službě Google Play.

Výsledná aplikace ale obsahuje i několik nedostatků. Záznam při maximální frekvenci GPS a senzorů značně snižuje výdrž baterie.

Nadmořská výška, kterou aplikace získává, neodpovídá té skutečné, kdy se rozdíl pohybuje kolem 50 metrů oproti skutečné výšce.

Aplikace umí komunikovat pouze s externím senzorem SensorTag CC2650. Je možné, že bude umět komunikovat i s jinými zařízeními typu SensorTag, ale tato podpora nebyla ověřena.

Aplikace neumí zobrazit data z GPS, která byla uložena spolu se záznamem. Pro jejich zobrazení je třeba exportovat uložený záznam a buď data přečíst přímo z GPX souboru, anebo použít aplikaci, která umí GPS data z GPX souboru zobrazit. Avšak tyto aplikace neumí zobrazit data ze senzorů, které byly pořízené na daných GPS pozicích.

Zmíněné nedostatky jsou oblasti, ve kterých je možné aplikaci v budoucnu zdokonalit spolu se zajištěním stálé podpory nejnovějších verzí Android.

Literatura

[1] Akcelerometr. How accelerometers work | Types of accelerometers [online]. [cit. 2018-04-08]. Dostupné z: http://www.explainthatstuff.com/accelerometers.html

[2] Senzory vlhkosti. Humidity Sensor: Basics, Usage, Parameters and Applications [online]. [cit. 2018-04-08]. Dostupné z: https://electronicsforu.com/resources/electronics-components/humidity-sensor-basic-usage-parameter

[3] Senzory tlaku. Pressure sensor - Sensor Technology - Metropolia Confluence [online]. [cit. 2018-04-08]. Dostupné z: https://wiki.metropolia.fi/display/sensor/Pressure+sensor

[4] Senzory okolního světla. WiseGEEK - What is a Light Sensor? (with pictures) [online]. [cit. 2018-04-08]. Dostupné z: http://www.wisegeek.org/what-is-a-light-sensor.htm

[5] Teplené senzory. Types of Temperature Sensors and Their working Principles | Features [online]. [cit. 2018-04-08]. Dostupné z: https://www.elprocus.com/temperature-sensors-types-working-operation/

[6] GAP. GAP | Introduction to Bluetooth Low Energy | Adafruit Learning System [online]. [cit. 2018-04-07]. Dostupné z: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap

[7] GATT. GATT | Introduction to Bluetooth Low Energy | Adafruit Learning System [online]. [cit. 2018-04-07]. Dostupné z: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

[8] CC2650 SensorTag. CC2650 SensorTag User's Guide - Texas Instruments Wiki [online]. [cit. 2018-02-21]. Dostupné z: http://processors.wiki.ti.com/index.php/CC2650_SensorTag_-Userś_Guide

[9] Sensortag CC2650. In: Texas Instruments SensorTag CC2650 (11) - IoTool [online]. [cit. 2018-02-21]. Dostupné z: https://iotool.io/extensions/sensors/texas-instruments-sensortag-cc2650-environmental-sensor-11

[10] Android Dashboards. Dashboards | Android Developers [online]. [cit. 2018-02-22]. Dostupné z: https://developer.android.com/about/dashboards/index.html

[11] Bluetooth Low Energy. Bluetooth Low Energy | Android Developers [online]. [cit. 2018-02-22]. Dostupné z: https://developer.android.com/guide/topics/connectivity/bluetooth-le.html

[12] SparseArray výhody a nevýhody. SparseArray vs HashMap in Android [online]. [cit. 2018-04-08]. Dostupné z: https://gunhansancar.com/sparsearray-vs-hashmap/

[13] SparseArray porovnání s HashMap. Android Performance Tweaking: ParseArray Versus Hashmap | Java Code Geeks - 2018 [online]. [cit. 2018-04-08]. Dostupné z: https://www.-javacodegeeks.com/2012/07/android-performance-tweaking-parsearray.html

[14] SparseArray porovnání s HashMap. Android: Should you use HashMap or SparseArray? - GREENSPECTOR® [online]. [cit. 2018-04-08]. Dostupné z: https://greenspector.com/-en/articles/2017-04-11-android-containers/

[15] Knihovna Androidplot. GitHub - halfhp/androidplot [online]. [cit. 2018-03-24]. Dostupné z: https://github.com/halfhp/androidplot

[16] Knihovna HelloCharts. GitHub - lecho/hellocharts-android [online]. [cit. 2018-03-24]. Dostupné z: https://github.com/lecho/hellocharts-android

[17] Knihovna MPAndroidChart. GitHub - PhilJay/MPAndroidChart [online]. [cit. 2018-03-24]. Dostupné z: https://github.com/PhilJay/MPAndroidChart

[18] OpenGL ES na Androidu. OpenGL ES | Android Developers [online]. [cit. 2018-03-26]. Dostupné z: https://developer.android.com/guide/topics/graphics/opengl.html

[19] SQLite. About SQLite [online]. [cit. 2018-03-26]. Dostupné z: https://www.sqlite.org/-about.html

[20] GPX. GPX: the GPS Exchange Format [online]. [cit. 2018-03-21]. Dostupné z: http://www.-topografix.com/gpx.asp

[21] Android Oreo změny. Android 8.0 Behavior Changes | Android Developers [online]. [cit. 2018-02-27]. Dostupné z: https://developer.android.com/about/versions/oreo/android-8.0-changes.html

[22] API pro práci se senzory. Android.hardware | Android Developers [online]. [cit. 2018-03-21]. Dostupné z: https://developer.android.com/reference/android/hardware/package-summary.html

[23] Datové typy SQLite. Datatypes In SQLite Version 3 [online]. [cit. 2018-02-27]. Dostupné z: https://www.sqlite.org/datatype3.html

[24] Sensors Multitool. Sensors Multitool - Android Apps on Google Play [online]. [cit. 2018-03-01]. Dostupné z: https://play.google.com/store/apps/details?id=com.wered.-sensorsmultitool

[25] Android Sensors. Android Sensors – Aplikace na Google Play [online]. [cit. 2018-03-30]. Dostupné z: https://play.google.com/store/apps/details?id=com.lpellis.sensorlab

[26] Sensor Sense. Sensor Sense - Android Apps on Google Play [online]. [cit. 2018-03-01]. Dostupné z: https://play.google.com/store/apps/details?id=com.kristofjannes.sensorsense

[27] Sensor Kinetics. Sensor Kinetics - Android Apps on Google Play [online]. [cit. 2018-03-01]. Dostupné z: https://play.google.com/store/apps/details?id=com.innoventions.sensorkinetics

[28] NRF Connect for Mobile. NRF Connect for Mobile - Android Apps on Google Play [online]. [cit. 2018-03-01]. Dostupné z: https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp

[29] Sensors Toolbox - SenseView. Sensors Toolbox - SenseView - Android Apps on Google Play [online]. [cit. 2018-03-01]. Dostupné z: https://play.google.com/store/apps/details?id=si.mobili.senseview

[30] Simplelink SensorTag. Simplelink SensorTag - SenseView - Android Apps on Google Play [online]. [cit. 2018-03-01]. Dostupné z: https://play.google.com/store/apps/-details?id=com.ti.ble.sensortag

A Přílohy na CD/DVD

Droidsor.apk - soubor pro instalaci aplikace

Droidsor - složka se zdrojovými kódy

ULI0012.pdf - textová část této bakalářské práce